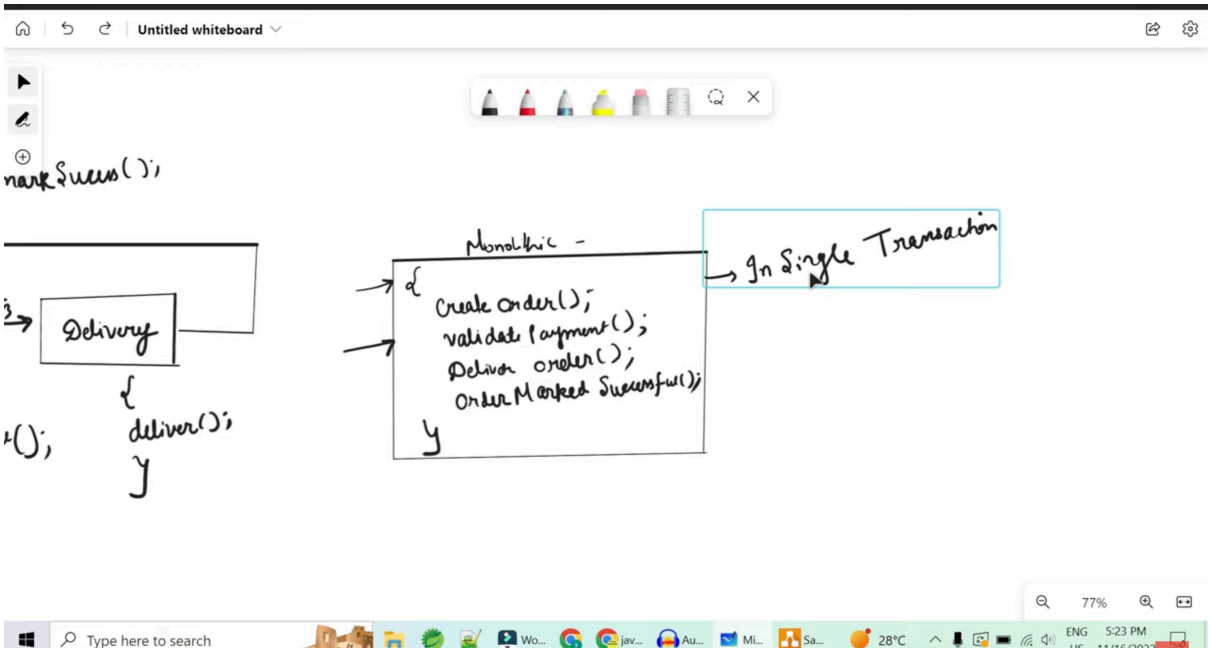
SAGA Design pattern
Microservices
Interview questions

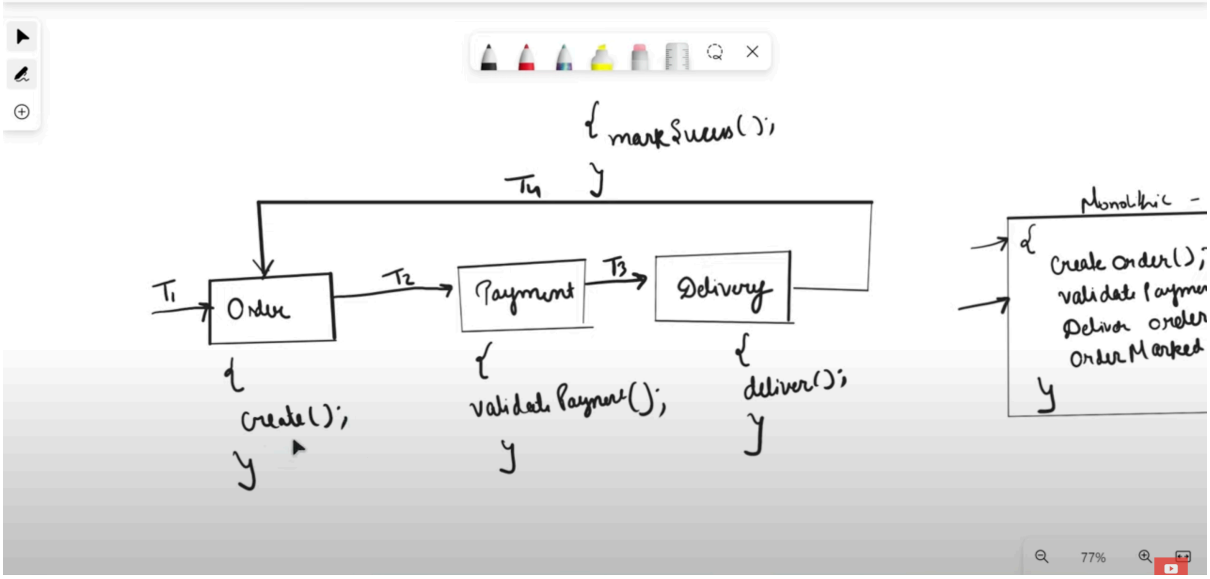
Why SAGA ??

- We know that Design pattern gives solutions to common problems faced by us “THE DEVELOPERS”. So **What problem is solved** by this SAGA design pattern ?
- The problem started as soon as we moved from Monolithic application to Microservice Architecture.
- We will take example of Swiggy , zomato.
- You
    - Choose your dishes,
    - Add them to Cart and checkout
    - Make Payment
    - Order gets Delivered
    - Our order is marked as completed after delivery is successful.
- In monolithic it’s not a problem as we have 1 database , multiple Tables like Orders, Payments, Delivery Etc. Now in 1 Single Atomic transaction we can do all these steps and if payment fails, everything gets rolled back.

Why SAGA ??

- Now we moved to microservices architecture and Segregated the whole zomato or swiggy application to
  - Order service
  - Payment service
  - Delivery Service
- Now your order service accepts your order, Payment service validates the payment done and Delivery service is responsible for delivery of your order to your home. When delivered successfully the orders is marked completed in the application. This is happy case.
- Ever thought about the worst case **Delivery is failed as no delivery partner was available.** Your payment was done, Money got deducted and now No food. At Least we need to get the money back and Order must be marked as cancelled.
- For this to happen we need a Transaction rollback . Transaction did get **rolled back** but only the **scope of transaction was in Delivery service**. **The boundary for this transaction ended in Delivery service**.





Why SAGA ??

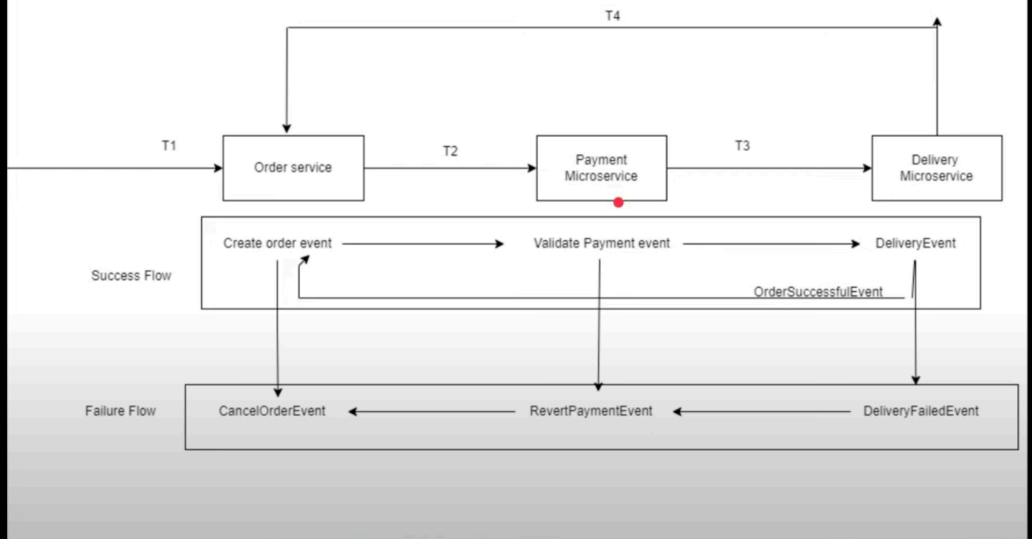
- Now what about the order service and payment service ?? **Neither your money is returned with this rollback nor your order status changed from waiting to failed / Cancelled.** Such a bad user experience right ?
- This is the classic example where your application completely failed to manage distributed transaction (A transaction what spans across multiple Microservices). Now This is a problem and To handle such distributed transactions issues SAGA Design pattern came into picture.

What is SAGA

- A saga is that sequence of local transactions.
- Each Saga has 2 Jobs to Do
  - Update the current Microservice and make required changes.
  - Publish events to trigger the next transaction for the next microservices.

How SAGA DP handles failure of any individual SAGA?

- The saga pattern provide**s transaction management** with using a sequence of local transactions of microservices. **Every microservices** has its own database and it can able to **manage local transaction in atomic way** with strict consistency.
- So saga pattern **grouping these local transactions and sequentially i**nvoking one by one. Each local transaction updates the database and **publishes an event to trigger the next local transaction.**
- If one of the step is failed, than saga patterns trigger to **rollback transactions** that are a set of **compensating transactions that rollback the changes on previous microservices** and restore data consistency.

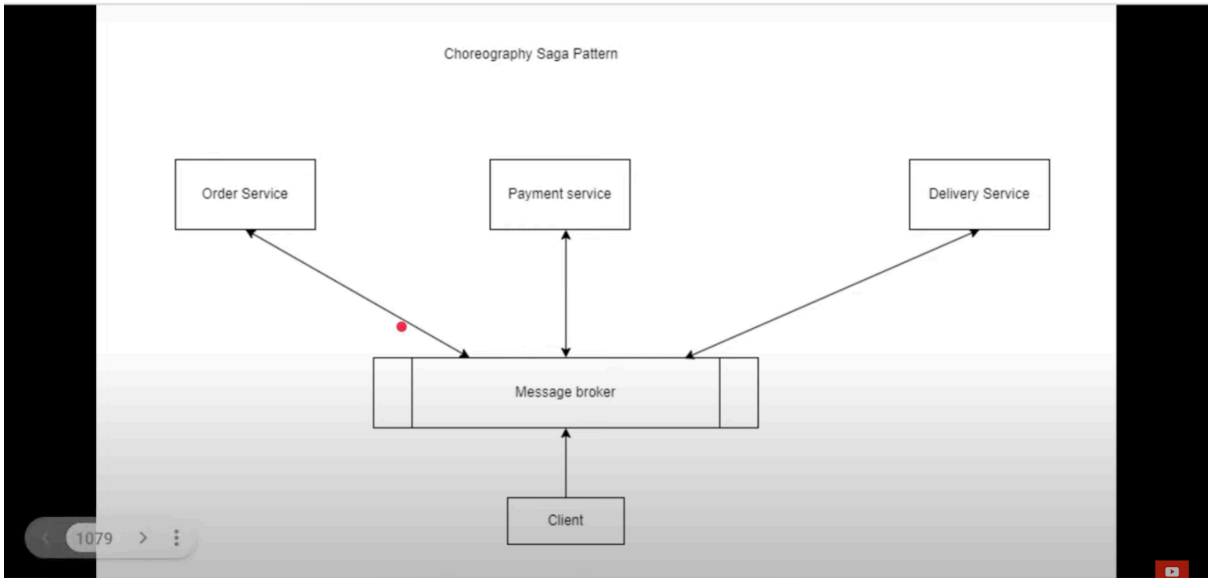


Ways to Implement SAGA?

- There are two type of saga implementation ways
  - choreography
  - orchestration

What is Choreography Saga Pattern?

- Choreography is a way to coordinate sagas where participants exchange events without a centralized point of control
- With choreography, each microservices run its own local transaction and publishes events to message broker system and that trigger local transactions in other microservices.



Advantages of Choreography Saga Pattern?

- Good for simple workflows that require few participants and don't need a coordination logic
- Doesn't require additional service implementation and maintenance.
- Doesn't introduce a single point of failure, since the responsibilities are distributed across the saga participants.

Disadvantages of Choreography Saga Pattern?

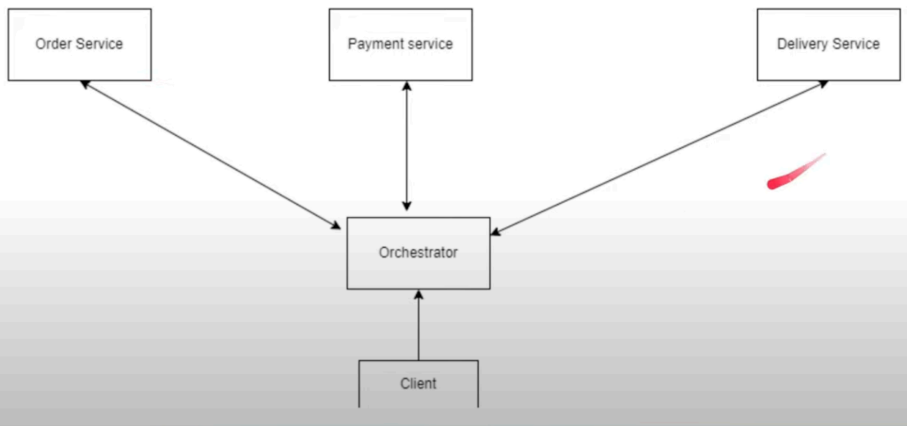
- Workflow can become confusing when adding new steps, as it's difficult to track which saga participants listen to which commands.
- There's a risk of cyclic dependency between saga participants because they have to consume each other's commands
- Integration testing is difficult because all services must be running to simulate a transaction.

What is Orchestration Saga Pattern?

- Orchestration is a way to **coordinate sagas where a centralized controller** tells the saga participants what local transactions to execute.
- The saga **orchestrator handles all the transactions** and **tells the participants which operation to perform based on events**.
- The orchestrator
  - executes saga requests,
  - stores and interprets the states of each task,
  - handles failure recovery with compensating transactions.

2

Orchestration Saga Pattern



Advantages of Orchestration Saga Pattern?

- Good for complex workflows involving many participants or new participants added over time.
- Suitable when there is control over every participant in the process, and control over the flow of activities.
- Doesn't introduce cyclic dependencies, because the orchestrator unilaterally depends on the saga participants.
- Saga participants don't need to know about commands for other participants. Clear separation of concerns simplifies business logic.

Disadvantages of Orchestration Saga Pattern?

- Additional design complexity requires an implementation of a coordination logic.
- There's an additional point of failure, because the orchestrator manages the complete workflow.